STRATEGY MANAGEMENT IN A MULTI-AGENT SYSTEM USING NEURAL NETWORKS FOR INDUCTIVE AND EXPERIENCE-BASED LEARNING

Florin LEON
Technical University „Gheorghe Asachi" of Iaşi, Str. Prof. dr. D. Mangeron no. 67, Iaşi, Romania
e-mail: florinleon@gmail.com

Andrei D. LECA
Technical University „Gheorghe Asachi" of Iaşi, Str. Prof. dr. D. Mangeron no. 67, Iaşi, Romania
e-mail: andy_leca@yahoo.com

Gabriela M. ATANASIU
Technical University „Gheorghe Asachi" of Iaşi, Str. Prof. dr. D. Mangeron no. 67, Iaşi, Romania
e-mail: gabriela.atanasiu@gmail.com

Management & Marketing
Challenges for Knowledge Society
(2010) Vol. 5, No. 4, pp. 3-28

Abstract. *Intelligent agents and multi-agent systems prove to be a promising paradigm for solving problems in a distributed, cooperative way. Neural networks are a classical solution for ensuring the learning ability of agents. In this paper, we analyse a multi-agent system where agents use different training algorithms and different topologies for their neural networks, which they use to solve classification and regression problems provided by a user. Out of the three training algorithms under investigation, Backpropagation, Quickprop and Rprop, the first demonstrates inferior performance to the other two when considered in isolation. However, by optimizing the strategy of accepting or rejecting tasks, Backpropagation agents succeed in outperforming the other types of agents in terms of the total utility gained. This strategy is learned also with a neural network, by processing the results of past experiences. Therefore, we show a way in which agents can use neural network models for both external purposes and internal ones.*

Keywords: agents, learning, neural networks, strategy management multi-agent system.

1. Introduction

The modern approach to artificial intelligence is centred on the concept of a rational agent (Russell and Norvig, 2002). An agent is a software or hardware entity that can perceive its environment through sensors and act upon that environment through actuators. An agent that always tries to optimize an appropriate performance measure is called a rational agent. Agents are seldom stand-alone systems, in many situations they coexist and interact with other agents in several different ways. A system that consists of a group of agents that can potentially interact with each other is called a multi-agent system, or MAS. This technology can be used to develop agents that act on behalf of a user and are able to negotiate with other agents in order to achieve their goals (Nikos, 2007). Auctions on the Internet and electronic commerce are such examples.

Agents and multi-agent systems are a natural way to conceptualize, understand, analyze, design, and implement complex and distributed systems from real world, composed of autonomous individual entities, which act and interact in an organized way. Agent and multi-agent oriented models are being used in several domains of applied research such as social simulation, robotics, user interfaces, computer-mediated collaboration, games, electronic business, information retrieval, education and training.

Some researchers believe that agents will become the next major computing paradigm and will be pervasive in every market (Janca, 1995). In many domains, intelligent agents must coordinate their activities in order to achieve both individual and collective goals: *Industrial Applications* (Process Control, Manufacturing, Air Traffic Control), *Commercial Applications* (Information Management, Electronic Commerce, Business Process Management), *Medical Applications* (Patient Monitoring, Health Care), and *Entertainment* (Games).

In the following we present some business applications based on intelligent agents, adapted after Jennings and Wooldridge (1998):

- *Information Management*

– Maxims application, an electronic mail filtering agent (Maes, 1994). The program learns to prioritize, delete, forward, sort, and archive mail messages on behalf of a user. It works by "looking over the shoulder" of a user as he/she works with his/her email reading program, and uses every action the user performs as a lesson;

– Zuno Digital Library (Ferguson and Wooldridge, 1997): A digital library is an organized, managed collection of data, together with services to assist the user in making use of this data. The Library is a multi-agent system that enables a user to obtain a single, coherent view of incoherent, disorganized data sources such as the World Wide Web, a user's own data, collections of articles on publishing house sites, and so on;

– Webdoggie (Lashkari, Metral and Maes, 1994): An intelligent agent created at MIT, to be used as a personalized World Wide Web document filtering

system. It recommends new World Wide Web documents based on documents the user liked in the past;

- *Electronic Commerce*. Commerce is almost entirely driven by human interactions; people make the decisions about when to buy goods, how much they should pay, etc. However, some commercial aspects can be automated and the decisions can be made by agents.

– Kasbah is a simple "electronic marketplace" (Chavez and Maes, 1996) with buying and selling agents for each good to be purchased or sold;

– BargainFinder (Crabtree and Jennings, 1996): An intelligent agent that searches music retailers on the Internet to locate and find the best price of audio CDs;

- *Business Process Management*. Managers usually make decisions based on a combination of judgement and information from many departments. Ideally, all relevant information should be gathered before taking a decision. For this reason, many organizations tried to develop a number of IT systems to assist with various aspects of the business processes management.

– Project ADEPT (Jennings et al., 1996) tries to solve this problem by viewing a business process as a community of negotiating, service-providing agents. Each agent represents a distinct role or department in an enterprise and is capable of providing one or more services.

2. The Architecture of the Multi-agent System

The multi-agent solution implemented using the JADE framework (Bellifemine, Caire and Greenwood, 2007) for solving general inductive learning problems is composed of six main classes of agents:

- The *Requester* is the agent that asks the *Solver* for a solution to an inductive learning problem;
- The *Solver* is the intermediary agent between a *Requester* and the solver agents. It receives a problem solving request, forwards it to the appropriate solver agent and then returns the solution, or an exception, to the *Requester* agent;
- Three concrete solvers: *Backpropagation*, *Quickprop*, and *Rprop* agents receive a problem name and parameters from the *Solver*, get the actual data from the *Environment*, and train a neural network with their respective algorithms to fit the data;
- The *Environment* knows the content of the inductive learning problem to be solved and transmits it to the concrete solver agents.

It is possible that more agents from a class should exist in the system. It is particularly common to have more requesters. The design also permits the existence of more *Solver* and *Environment* agents (Leon, 2010).

Complying with FIPA recommendations, the *Solver* and *Environment* agents first register their services in the JADE "yellow pages" or "directory facilitator". In

this way, an *Environment* agent must be found by a *Solver* agent, and a *Solver* agent must be found by a *Requester*. The typical sequence of solving a problem is presented in figure 1.

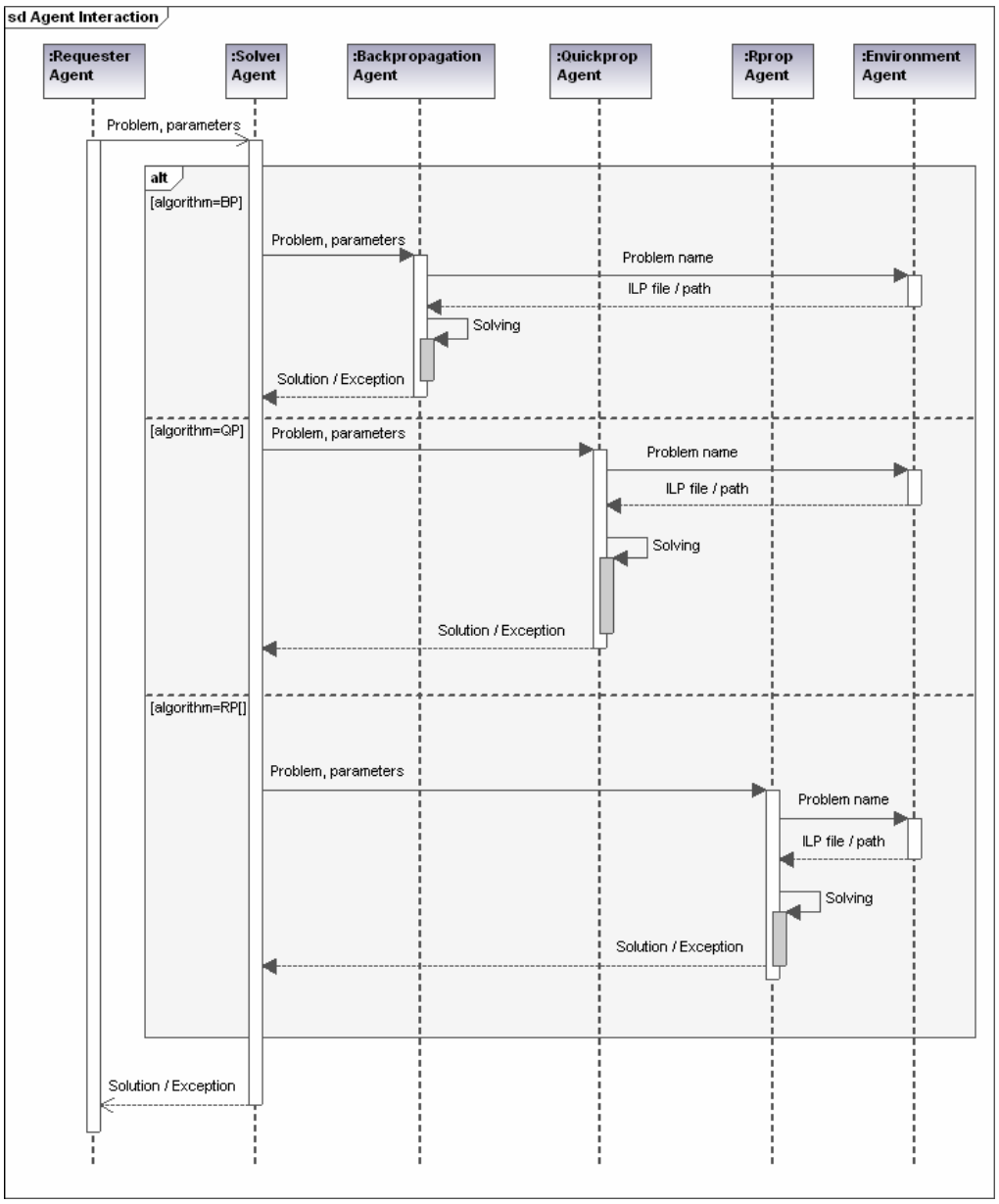

Figure 1. UML sequence diagram of agent interaction

The actual processing of the agents is implemented with the help of JADE *Behaviour* classes. All the agents have a "cyclic" behaviour for receiving messages from other agents. The JADE environment manages a message queue for all the agents. When an agent decides to receive a message, the first message in its queue becomes available. An asynchronous message handling approach is used. The agents continuously wait for messages and when one message becomes available, it is handled following the protocol of each class of agents, as presented in Figure 1.

3. Solving Inductive Learning Problems

3.1. Classification and Regression

Classification is a procedure in which individual items or instances are placed into groups, or "classes", based on information about their characteristics, referred to as attributes. Classification is a supervised technique, i.e. the model is built based on a training set of instances whose classes are known. Formally, given the training data $\{(\vec{x}_1, y_1),...,(\vec{x}_n, y_n)\}$, $\vec{x}_i \in X$ and $y_i \in Y$, the goal is to determine a classifier $h: X - Y$ which is a good approximation of the mapping of any object $\vec{x}_i$ to its classification label, or "class" y_i. More simply, the purpose of a classification algorithm is to find a hypothesis h from examples, i.e. pairs $(\vec{x}, f(\vec{x}))$, where f is a function with a discrete codomain, such that $h \approx f$ holds not only for the given training pairs, but also for other arguments $\vec{x}$ from their domain. Thus the information contained in the training set (with instances whose corresponding class labels are known) is used to classify new, previously unseen instances based on an explicit model for "eager" learners such as the decision trees, or an implicit model for "lazy" learners such as the instance-based methods (Atanasiu, Leon and Popa, 2008; Leon, Lisa and Curteanu, 2010).

Regression or function approximation is similar in principle to classification, with the main difference that the output is not discrete, as it is the case with classification, but continuous and real-valued.

3.2. Neural Networks. Training algorithms

Feed-forward neural networks are a method for building models usually containing non-linear relations. The processing elements of a network, the neurons, are organized in layers and each neuron is linked to the neurons of the next layer. Typically, a feed-forward network consists of one input layer, some hidden layers and an output layer. In the training phase, the neural network learns the mapping between the inputs and the output(s) by trying to find the values of the connection weights that minimize the differences between the actual network outputs and the desired values. The purpose of developing a neural model is to devise a network that captures the

essential relationships in the data. Then it can be applied to new sets of inputs to produce corresponding outputs. This is called generalization and represents the subsequent phase after training, i.e. the testing phase. A network is said to generalize well when the input-output relationship found by the network is correct for the input/output patterns of test data that were not used for the training of the network.

3.2.1. Backpropagation

Backpropagation algorithm (Bryson and Ho, 1969; Werbos, 1974; Rumelhart, Hinton and Williams, 1986) is one of the best-known algorithms for training neural networks. The forward pass produces an output vector for a given input vector based on the current state of the network weights. Since the network weights are initialized to random values, it is unlikely that reasonable outputs will result before training. The weights are adjusted to reduce the error by propagating the output error backward through the network. This process is where the algorithm gets its name from and is known as the backward pass:

- Compute error values for each node in the output layer. This can be computed because the desired output for each node is known;
- Compute the error for the middle layer nodes. This is done by attributing a portion of the error at each output layer node to the middle layer node, which feed that output node. The amount of error due to each middle layer node depends on the size of the weight assigned to the connection between the two nodes;
- Adjust the weight values to improve network performance using the updating weights method;
- Compute the overall error to test network performance.

The training set is repeatedly presented to the network and the weight values are adjusted until the overall error is below a predetermined tolerance.

The formula for updating weights in output layer weights is given by the following equation:

$$\Delta w_{kj} = \eta(t_k - o_k)(1 - o_k)o_k y_j, \quad (1)$$

and the change in hidden layer weights is propagated back using the formula below:

$$\Delta v_{kj} = z_i \sum_{k=1}^{K} (1 - y_j) w_{kj} \Delta v_{kj} \quad (2)$$

Where t_k is the target for the k^{th} output neuron, o_k is the current calculated value for the k^{th} output neuron, η is the learning rate. A momentum term which preserves the velocity of weight updates is specified by α.

The pseudocode of the algorithm is as follows (Peer, 2005):

Initialise v_{ji}, w_{kj} with small random values

t = 0

repeat
  for all traning pattern **do**
    $w_{kj}=w_{kj} + \Delta w_{kj}(t) + \alpha\Delta w_{kj}(t-1)$
    $v_{ji}= v_{ji}+\Delta v_{ji}(t) + \alpha\Delta v_{ji}(t-1)$
  end for
  t=t+1
until (stopping condition)

3.2.2. Quickprop

The quickprop algorithm (Fahlman, 1988) is a modification of the standard backpropagation algorithm that uses an approximation to the error curve in order to speed up the computation. The formula for updating the weights is:

$$\Delta w(t) = -\eta\delta E/\delta w(t) + \mu\Delta w(t-1), \quad (3)$$

where $\delta E/\delta w(t)$ is the error derivative for that weight accumulated over the whole epoch, η is learning rate and μ is maximum growth factor.

The pseudocode of the algorithm is as follows (Srinivasan, 2008):

repeat
  for all weight w_i
    if $\Delta w_{i-1} > 0$ **then**
      if $\delta E/\delta w_i > 0$ **then**
        $\Delta w_i = \eta\delta E/\delta w_i$
      if $\delta E/\delta w_i > (\mu/1+ \mu)\ \eta\delta E/\delta w_{i-1}$ **then**
        $\Delta w_i = \Delta w_{i+} \mu\Delta w_{i-1}$
      else
        $\Delta w_i = \Delta w_i + (\delta E/\delta w_i * \Delta w_{i-1}) / (\delta E/\delta w_{i-1} - \delta E/\delta w_i)$
    else if $\Delta w_{i-1} < 0$ **then**
      if $\delta E/\delta w_i < 0$ **then**
        $\Delta w_i = \eta\delta E/\delta w_i$
      if $\delta E/\delta w_i < (\mu/1+ \mu)\ \eta\delta E/\delta w_{i-1}$ **then**
        $\Delta w_i = \Delta w_{i+} \mu\Delta w_{i-1}$
      else
        $\Delta w_i = \Delta w_i + (\delta E/\delta w_i * \Delta w_{i-1}) / (\delta E/\delta w_{i-1} - \delta E/\delta w_i)$
    else
      $\Delta w_i = \eta\delta E/\delta w_i$
    $w_i = w_i + \Delta w_i$
  end for
until (stopping condition)

3.2.1. Resilient Propagation

Resilient backpropagation, Rprop, is a local adaptive learning scheme, performing supervised batch learning in multi-layer perceptrons (Riedmiller and Braun, 1993). The basic principle of Rprop is to eliminate the harmful influence of the size of the partial derivative on the weight step. As a consequence, only the sign of the derivative is considered to indicate the direction of weight update. The size of the weight change is exclusively determined by a weight-specific "update value" $\Delta_{ij}^{(t)}$:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)}, \frac{\partial E}{\partial w_{ij}}^{(t)} < 0, \\ 0, \textit{altfel} \end{cases} \quad (4)$$

where $\frac{\partial E}{\partial w_{ij}}^{(t)}$ denotes the summed gradient information over all patterns of the pattern set.

The second step of Rprop learning is to determine the new update values $\Delta_{ij}(t)$. This is based on a sign-dependent adaption process.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} \cdot \Delta_{ij}^{(t-1)}, \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-} \cdot \Delta_{ij}^{(t-1)}, \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0, \\ \Delta_{ij}^{(t-1)}, \textit{altfel} \end{cases} \quad (5)$$

where $0 < \eta^{-} < 1 < \eta^{+}$

The pseudocode of the algorithm is as follows (Riedmiller, 1994):

Initialise

$\forall i, j : \Delta_{ij}(t) = \Delta_0$

$\forall i, j : \frac{\partial E}{\partial w_{ij}}(t-1) = 0$

repeat

Calculate gradient $\frac{\partial E}{\partial w}(t)$

for all weight and biases:

if $(\frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)>0)$ **then**

$$\Delta_{ij}(t)=\min(\Delta_{ij}(t-1)\cdot\eta^{+},\Delta_{\max})$$

$$\Delta w_{ij}(t)=-\operatorname{sgn}(\frac{\partial E}{\partial w_{ij}}(t))\cdot\Delta_{ij}(t)$$

$$w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)$$

$$\frac{\partial E}{\partial w_{ij}}(t-1)=\frac{\partial E}{\partial w_{ij}}(t)$$

else if $(\frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)<0)$ **then**

$$\Delta_{ij}(t)=\max(\Delta_{ij}(t-1)\cdot\eta^{-},\Delta_{\min})$$

$$\frac{\partial E}{\partial w_{ij}}(t-1)=0$$

else if $(\frac{\partial E}{\partial w_{ij}}(t-1)*\frac{\partial E}{\partial w_{ij}}(t)=0)$ **then**

$$\Delta w_{ij}(t)=-\operatorname{sgn}(\frac{\partial E}{\partial w_{ij}}(t))\cdot\Delta_{ij}(t)$$

$$w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)$$

$$\frac{\partial E}{\partial w_{ij}}(t-1)=\frac{\partial E}{\partial w_{ij}}(t)$$

until (stopping condition)

In this pseudocode, Δ_0 is the initial update value (default settings value is 0.1, the choice of this value is rather uncritical), $\Delta_{\max}$ is the maximum weight step (set somewhat arbitrary to 50), and $\Delta_{\min}$ is the minimum step size (constantly fixed to10^{-6}).

3.3. Task Allocation

In order to study the behaviour of the multi-agent system where the agents use neural networks to solve classification and regression tasks, we considered 20 typical problems. They are briefly described as follows.

The simplest class classification problems are the Boolean logic ones, such as AND or XOR (exclusive or). Although they have an equal number of attributes and

instances, the main difference between them is that the former is a linearly separable problem (the instances of the two classes can be separated into two distinct semiplanes) and the later is not, therefore it is more difficult to learn.

Table 1

Boolean logic functions: AND and XOR

x	y	AND	XOR
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	false

The Iris classification problem is perhaps the best-known to be found in the literature. Fisher's paper (1936) is a classic in the field and is frequently referenced (Duda and Hart, 1973). The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant: *Setosa*, *Versicolor*, and *Virginica*. The *Setosa* class is linearly separable from the other two, which in turn are not linearly separable from each other. There are 4 numeric attributes: petal length, petal width, sepal length, and sepal width, all expressed in centimetres. There are no missing attributes.

For the multi-agent system analysis, we considered the original numerical version of the problem, but also 2 other versions where the numerical attributes are discretized into 3 and 5 values, respectively.

The three MONK's problems are another benchmark example on which extensive studies have been performed (Thrun, et al., 1991). They assume that a robot is to be described by 6 attributes: head shape (round, square, octagon), body shape (round, square, octagon), whether it is smiling (yes, no), the object it is holding (sword, balloon, flag), jacket colour (red, yellow, green, blue), and whether it has a tie (yes, no). The learning task is a binary classification one. Each problem is given by a logical description of a class. Robots belong either to this class or not, but instead of providing a complete class description to the learning problem, only a subset of 432 possible robots is given. The classifier must generalize over a rather small subset of these 432 examples, and predict the class membership for the instances left out.

Problem 1 is defined as: *(head shape = body shape) or (jacket colour is red)*. From 432 examples, 124 were randomly selected for training, without misclassifications. The problem is in standard disjunctive normal form and should be easily learnable by symbolic learning algorithms.

Problem 2 is: *exactly two of the six attributes have their first value*. From 432 examples, 169 were randomly selected for training, without misclassifications. The problem is similar to parity problems and it combines different attributes in a way that makes it difficult to be described in disjunctive or conjunctive normal form using only the given attributes.

Problem 3 is defined as: *(jacket colour is green and holding a sword) or (jacket colour is not blue and body shape is not octagon)*. From 432 examples, 122 were randomly selected for training, with 5% misclassifications. The problem is also

in disjunctive normal form and serves to evaluate the algorithms under the presence of noise.

Another simple problem is the Drugs problem that tries to find the rules to prescribe one of the two classes of drugs, according to the patient's age, blood pressure, and gender. The age attribute is numeric, the other two are nominal: blood pressure may be low, normal or high, and the gender may be female or male. A good classifier should discover that the gender attribute is irrelevant to the categorization problem. If we consider that there are two drug types: *A* and *B*, the instances are presented in Table 2.

Table 2

The drugs problem

Gender	Age	Blood pressure	Drug
male	20	normal	A
female	73	normal	B
female	37	high	A
male	33	low	B
female	48	high	A
male	29	normal	A
female	52	normal	B
male	42	low	B
male	61	normal	B
female	30	normal	A
female	26	low	B
male	54	high	A

Shepard, Hovland and Jenkins (1961) studied human performance on a category learning task involving eight stimuli divided evenly between two categories. The stimuli were generated by varying exhaustively three binary dimensions. They observed that if these dimensions are regarded as interchangeable, there are only six possible category structures across the stimulus set (Navarro, Myung and Pitt, 2004), as shown in Table 3.

Table 3

Shepard's problems

Stimulus	Attributes			Class					
				1	2	3	4	5	6
1	0	0	0	A	A	A	A	A	A
2	0	0	1	A	A	A	A	A	B
3	0	1	0	A	B	A	A	A	B
4	0	1	1	A	B	B	B	B	A
5	1	0	0	B	B	B	A	B	B
6	1	0	1	B	B	A	B	B	A
7	1	1	0	B	A	B	B	B	A
8	1	1	1	B	A	B	B	A	B

In the first problem only the first attribute is significant for the categorization. The second problem is a XOR-type one. Problems 3, 4, and 5 are rule-plus-exception problems (that is why we tested only problem 4, since the other two are similar). Problem 6 compels the subject to memorize all stimuli, because there is no rule to group them.

A thorough comparison in terms of classification accuracy and execution time between several well-known algorithms is presented by Leon, Zaharia and Gâlea (2004).

Finally, a decision problem of whether to play golf based on the state of the weather on Saturday morning (Quinlan, 1986) was considered. It has 4 attributes and a binary class. There are two versions for the problem, one containing both numerical and symbolical attributes, and another with only symbolic attributes. The two versions are displayed together in table. The gray italic columns contain the alternative values of the *Temperature* and *Humidity* attributes.

Table 4

The "Weather" or "Playing golf" problem

Outlook	Temperature	*Temperature*	Humidity	*Humidity*	Windy	Play
sunny	85	*hot*	85	*high*	false	no
sunny	80	*hot*	90	*high*	true	no
overcast	83	*hot*	86	*high*	false	yes
rainy	70	*mild*	96	*high*	false	yes
rainy	68	*cool*	80	*normal*	false	yes
rainy	65	*cool*	70	*normal*	true	no
overcast	64	*cool*	65	*normal*	true	yes
sunny	72	*mild*	95	*high*	false	no
sunny	69	*cool*	70	*normal*	false	yes
rainy	75	*mild*	80	*normal*	false	yes
sunny	75	*mild*	70	*normal*	true	yes
overcast	72	*mild*	90	*high*	true	yes
overcast	81	*hot*	75	*normal*	false	yes
rainy	71	*mild*	91	*high*	true	no

The simplest regression problems considered are the maximum and sinus functions, with 2 attributes and 1 attribute, respectively. Beside them, we devised 3 original regression problems, based on the following equations.

Problem 1 is defined as:

$$f(x,y,z) = \sin(x)\cdot\cos(y) + \sqrt{|z|}, \tag{6}$$

where $x, y \in [-\pi, \pi]$ and $z \in [-2, 2]$.

Problem 2 is defined as:

$$f(x,y) = \sin(x)\cdot\cos(y) + \frac{\sin(y)}{2+\cos(x)}. \tag{7}$$

Problem 3 is defined as:

$$f(x,y) = \sin(x) + \cos(x + y). \quad (8)$$

The definition domains of the x and y arguments are the same as for the first problem.

3.4. The ILP File Format

All the inductive learning problems are different, but their structure is similar nevertheless: they all have a number of attributes of some type and instances defined by the values of these attributes. Therefore, there is a need to devise a file format able to describe a variety of problems in a consistent way. Such a file should also be able to be used by the great diversity of algorithms available today.

A well-known format is the so-called "Attribute-Relation File Format", or ARFF, developed to be used with the popular data mining software WEKA, Waikato Environment for Knowledge Analysis (Frank et al., 2008).

While WEKA allows the user to experiment with data in a straightforward manner, there are some characteristics of learning problems that are not included in the ARFF format. In order to address some of these issues, Leon proposes the ILP (Inductive Learning Problem) format. The differences from the ARFF format are discussed as follows.

1. The ARFF file format has two main sections: one which defines the attributes, and one which defines data, i.e. the actual instances. The class is not specially marked, it is usually considered to be the last attribute. The ILP file format explicitly distinguishes between the inputs and the outputs, especially because a problem in the general case could have more outputs. ILP is designed to be used by symbolic classification algorithms and also by sub-symbolic techniques, such as the neural networks. While it is possible to establish one separate model for each output, in some cases, e.g. neural networks, the user may want one network to approximate a vector function with multiple outputs. This case is facilitated by the distinction that ILP format makes;

2. ARFF assumes that the attributes are either numeric ("real") or symbolic, in which case the possible attribute values are listed in the definition of the attribute. ILP makes a finer distinction between the types of the attributes, as specified by the data mining and data analysis theory (Tan, Steinbach and Kumar, 2005): 2 numeric types (interval and ratio) and 2 symbolic types (nominal and ordinal). The difference between the nominal and the ordinal types is especially important for instance-based algorithms that compute the distance between instances. If no distinction is made, the distance between any two different values of an attribute would be considered to be 1. If an ordering relation is present, then in the case of a ranking with 5 items, such as: *Very Low*, *Low*, *Medium*, *High*, *Very High*, the difference between *Medium* and *High* could be 1/4 instead of 1, which could affect the results of the algorithm;

3. The ARFF format simply marks as "real" the continuous attributes. The ILP format allows the user to define the range of values for an interval or a ratio attribute. The actual range can be different from the range computed from the data. For example, when dealing with a trigonometric function, the actual range of the results should be [-1, 1]. If the data sampling is not fine enough, the computed range can be smaller. These differences in range can be important when the data are normalized for instance-based methods or neural networks;

4. A small but sometimes important difference is that the instance values are separated by comma in ARFF and by blank spaces or tabs in ILP. The tabs increase the quality of the display by aligning the attribute values; however there is a more significant reason for using blanks as separators. When the decimal separators are different, for example the English standard compared to the Romanian standards, where the comma and the dot have opposite meaning, the ILP format can help to avoid such problems;

5. A final difference from ARFF is that ILP uses 3 sections for the definition of data. The first section is called "Training" and it is used to build the model. The second, optional, is "Testing", used to validate the model and estimate its generalization capability. The third, again optional, is "Prediction", which contains the instance for which we do not know the class or the outputs, and want to apply the model in order to find them. These instances only have the input values defined. Even if the WEKA software allows the user to divide data into training and testing sets, this is not explicitly stated by the ARFF problem file.

The "Weather" or "Playing golf" problem described in paragraph 3.3 in ILP format is given below. The reserved words are marked with bold.

```
INPUT
ORDINAL Outlook : sunny overcast rainy
INTERVAL Temperature : 64 85
RATIO Humidity : 65 96
NOMINAL Windy : false true

OUTPUT
NOMINAL Play : no yes

TRAINING
sunny 85    85    false no
...
```

The Iris problem has the following structure, including the "Testing" and "Prediction" sections:

```
INPUT
RATIO SepalLength : 4.3 7.9
RATIO SepalWidth : 2 4.4
RATIO PetalLength : 1 6.9
RATIO PetalWidth : 0.1 2.5
```

```
OUTPUT
NOMINAL Class : setosa, virginica, versicolor

TRAINING
5.1   3.5   1.4   0.2   setosa
...
5.5   2.4   3.7   1.0   versicolor
...
5.9   3.0   5.1   1.8   virginica

TESTING
5.7   2.9   4.2   1.3   versicolor
...

PREDICTION
5.7   2.8   4.1   1.3
...
```

The ILP model can be easily extended to include unknown values (marked as "?") and sparse data, by retaining only the non-default values.

4. Behaviour Analysis of the Multi-agent System

The multi-agent system under analysis, implemented within the JADE framework, is comprised of 15 agents that use three different training algorithms for their neural networks: 5 agents use Backpropagation, 5 agents use Quickprop, and 5 agents use Rprop.

The agents are given 500 tasks, consisting in repeated versions of the 20 classification and regression problems described in paragraph 3.3. The characteristics of these inductive learning problems are presented in table 5: the number of attributes, the type of the problem (classification – C or regression – R), the number of training instances, the number of testing instances, and the number of times the problem appears in the 500 tasks. Problems appear in random order.

Before executing the multi-agent system, the problems were tested separately, in order to estimate their complexity in terms of execution times while running the three training algorithms mentioned above.

Since finding the optimal topology of a neural network is a difficult problem, the agents successively try increasingly complex network configurations. The performance of the algorithms depend on the MSE (Mean Square Error) desired by the user, and also on the number of training epochs. In this scenario, the agents need to find a MSE of 0.001 or lower, and run the algorithms for 500 epochs. The MSE can be higher for classification problems, if the percent of correctly classified instances is 100% for both training and testing.

Table 5

The characteristics of the inductive learning problems

No	Name	Attributes	Type	Training Instances	Testing Instances	Number
01	and-sym	2	C	4	4	32
02	drugs	3	C	12	0	33
03	iris-d3	4	C	150	0	17
04	iris-d5	4	C	150	0	17
05	iris-num	4	C	150	12	17
06	max	2	R	6	3	33
07	monks1	6	C	124	432	25
08	monks2	6	C	169	432	17
09	monks3	6	C	122	432	17
10	problem1	3	R	1352	0	4
11	problem2	2	R	676	0	8
12	problem3	2	R	169	0	17
13	shepard1	3	C	8	0	32
14	shepard2	3	C	8	0	33
15	shepard4	3	C	8	0	33
16	shepard6	3	C	8	0	33
17	sin	1	R	40	0	33
18	weather	4	C	14	14	33
19	weather-sym	4	C	14	0	33
20	xor-sym	2	C	4	4	33

They first use a network topology of 1 hidden layer with 10 neurons. If this configuration proves to be too simple to learn the model, and the performance criteria cannot be met, they use a network topology of 1 hidden layer with 20 neurons. The third attempt is a topology of 2 hidden layers with 15 and 5 neurons, respectively. Finally, they use a topology of 2 hidden layers with 30 and 10 neurons, respectively.

The ratio between the number of neurons in the first and the second hidden layer follows Kudrycki's heuristic, with recommends a ratio of 3:1 (Kudrycki, 1988).

The number of epochs is rather small and the target MSE is rather large. However, these values can emphasise the difference in convergence speed of the training algorithms.

Table 6 displays the average values of 10 tests for each problem and training algorithm. The tests were made on a computer with Intel Pentium processor with a frequency of 3GHz, and 1 GB of RAM.

The multi-agent system assumes the following rules for task allocation. Tasks are proposed to the agents and they have the freedom of accepting a task or not. Only the available agents, i.e. agents that are not currently engaged in solving some other task can bid for a new one. Out of the bidding agents, the system randomly selects a maximum of 3 agents to solve a task. This redundancy is useful because the neural network training depends on the initial, random initialization of connection weights.

Therefore, a network can get stuck in a local minimum. However, the 4 successive attempts to solve the problem in addition to a maximum of 3 agents to solve the same problem should be enough to find a reasonable solution.

Table 6

Average execution time

No	Name	Average Time Backpropagation	Average Time Quickprop	Average Time Rprop
01	and-sym	3.9	1.5	1.4
02	drugs	3.5	2.4	2.1
03	iris-d3	20,540.1	20,512.1	20,603.9
04	iris-d5	21,037.9	21,195.9	21,216.6
05	iris-num	20,507.9	16,808.5	5,405.6
06	max	175.1	42.3	18.1
07	monks1	11,046.6	4,894.2	4,055
08	monks2	20,569.1	11,254.6	13,803
09	monks3	17,740.1	5,687.3	5,462.1
10	problem1	166,606.7	163,786.6	160,808.8
11	problem2	78,574.4	78,496.8	78,353.6
12	problem3	19615	19719	10139.9
13	shepard1	1.7	1.3	1.5
14	shepard2	1060.1	7.8	3.5
15	shepard4	1.7	1.2	1
16	shepard6	1057.8	8.9	3.4
17	sin	1094.6	1808.1	542.6
18	weather	50.6	29.5	20.4
19	weather-sym	63	22.4	15.8
20	xor-sym	552.1	4.4	1.5

If the system has less than 3 available agents, the task can be assigned to only 1 of 2 agents. If there are no agents available, the task allocation system waits until some agents become available again.

Agents do not know the task complexity beforehand. They can only have descriptive quantitative information such as that contained in columns 3 to 6 in Table 5. However, task with similar structure may have completely different difficulties, depending on the internal model, which is not known. Agents are also aware of the number of other available agents in the system (possible competitors), but they do not know the type of training algorithm the others are using.

The agents receive rewards for solving tasks. However, only the agent that provided the smallest error out of the maximum 3 competitors receives this reward. If more agents have the same error, the network with the smallest topology, and then the agent with the smallest execution time is declared the winner.

The utility that agents receive for solving a task is computed by taking into account the classification accuracy of the MSE of a regression problem.

For classification problems, the utility is given in terms of the percent of incorrectly classified instances:

$$U^C = 100 - P_{ICI} \quad (4)$$

If the problem has both training and testing data, then the utility is given as a weighted sum between the percent of incorrectly classified instances for training and for testing, by emphasising the importance of the generalization:

$$U^C = 100 - \frac{\left(P_{ICI}^{training} + 2 \cdot P_{ICI}^{testing}\right)}{3} \quad (4)$$

For regression problems, the received utility formula takes into account the ratio between the training MSE achieved by the agent and the maximum allowed error (in our case 0.001):

$$R = \frac{MSE_{training}}{MSE_{max}}. \quad (5)$$

A formula was devised for computing the utility in this case, by using the Multiple Multiplicative Model (MMF):

$$U_d = \frac{a \cdot b + c \cdot R^d}{b + R^d}, \quad (6)$$

with the following coefficient data: a = –12.912808, b = 35.995555, c = 464.803706, d = 0.36732278. The decrease in utility U_d is graphically displayed in Figure 2.

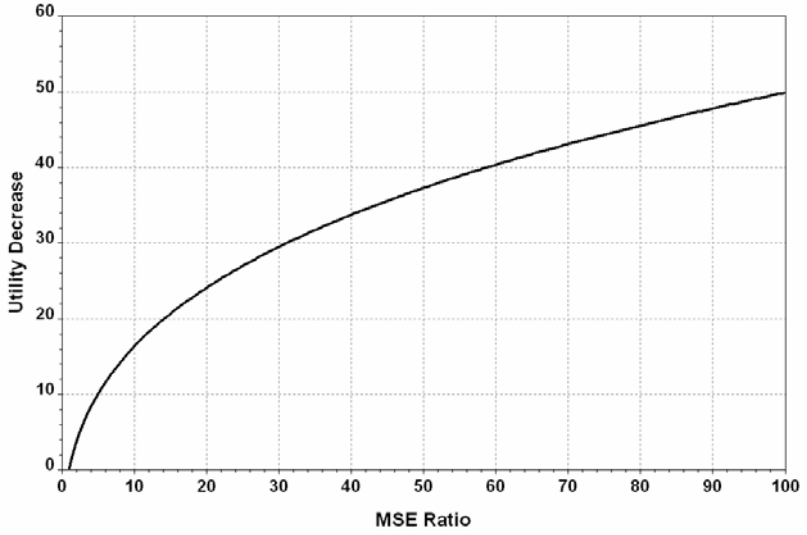

Figure 2. The decrease in utility as a function of the MSE ratio

Finally, the utility of solution for a regression problem is given by the following equation:

$$U^R = 100 - U_d \,. \tag{7}$$

If the problem also has testing data, the formula becomes:

$$U^R = 100 - \frac{\left(U_d^{training} + 2 \cdot U_d^{testing}\right)}{3} \,. \tag{8}$$

4.1. Reference Behaviour

In order to analyse and later optimize the utilities received by the agents, we established a reference result, when all the agents accept any given task, provided that they are available. The total utilities received by the agents are displayed in Figure 3. The last 3 bars represent the average utility received by each agent type: Backpropagation agents, Quickprop agents, and Rprop agents.

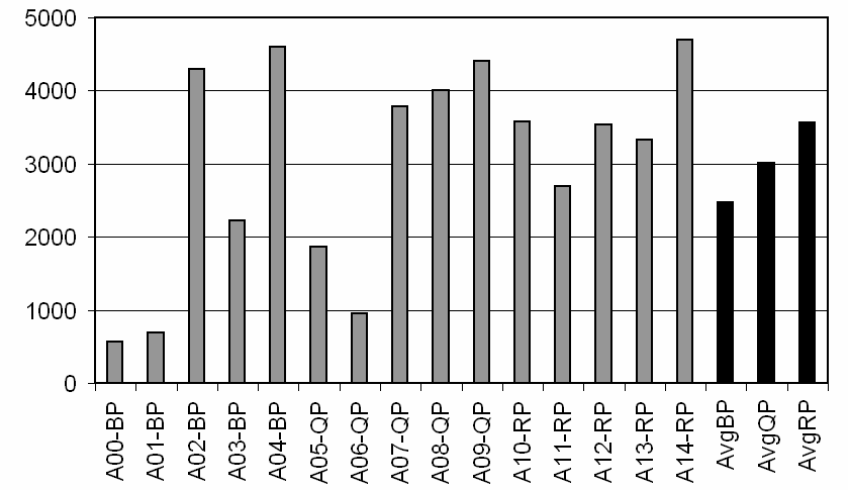

Figure 3. Reference behaviour: agents accept any task

4.2. Optimizing the Task Acceptance Strategy

It can be noticed in Figure 3 that Backpropagation agents perform poorly compared to the other agents. Quickprop agents have good results and Rprop agents perform the best. We aim to improve the performance of Backpropagation agents, and analyse how a change in their utility affects the other agents in the system. The strategy that we consider is based on task acceptance. For example, intuitively, the Backpropagation algorithm seems to be slower that the other two. Therefore, it would be logical for

Backpropagation agents to refuse the most complex, time-consuming tasks, that would take long to execute and even then, the error could be worse than that of their competitors and therefore no utility would be received. By getting more simple tasks more often, one could hope to improve the total utility received. Also, if there are more available agents in the system, there are more chances that they should be Quickprop or Rprop agents, which usually provide better solutions. Thus, if fewer agents are available, there are more chances that a Backpropagation agent should be the winner of a task utility.

4.2.1. Heuristics Based on Symbolic Models

We first try to optimize the strategy of agent A00, which has the worst results in the reference. In this respect, we use however the execution history of all Backpropagation agents, since they are homogeneous. We note the task information and the final result of accepting it: winning a reward or losing it. This represents another inductive learning problem, built out of the internal experience of the agents and not given by an external user.

Several approaches were tried. First, symbolic classification algorithms were used to extract explicit behavioural rules with the help of the C4.5 decision tree algorithm.

The C4.5 algorithm generates a decision tree for a given dataset by recursive partitioning of data (Quinlan, 1993; Joshi, 1997). The algorithm considers all the possible tests that can split the data set and selects a test that gives the best information gain. For each discrete attribute, one test with as many outcomes as the number of distinct values of the attribute is considered. For each continuous attribute, binary tests involving every distinct value of the attribute are considered. In order to gather the entropy gain of all these binary tests efficiently, the training data set belonging to the node in consideration is sorted for the values of the continuous attribute and the entropy gains of the binary cut based on each distinct value are calculated in one scan of the sorted data. This process is repeated for each continuous attribute.

This attempt proved that the resulting model was very complex. Neither of the two methods was able to learn it well. The main reason for this is that agents actually have imperfect information about the real model of the "world" (their execution environment). Since they cannot know the nature of a task, they can only use circumstantial information hoping that it would be correlated to the actual difficulty of the task.

The C4.5 algorithm builds a decision tree with a classification error of almost 17%. However, from the tree, there is some information that the tasks with more than 169 training instances and more than 4 attributes are usually lost.

Therefore, a rejection rule was added for the A00 agent, according to these findings. The resulting utilities of the agents are presented in Figure 4. One can see

that this simple rule has dramatically increased the performance of A00 agent. Since more tasks were won by A00, this also affected the direct competitors: the Quickprop agents, whose average performance decrease below 3000.

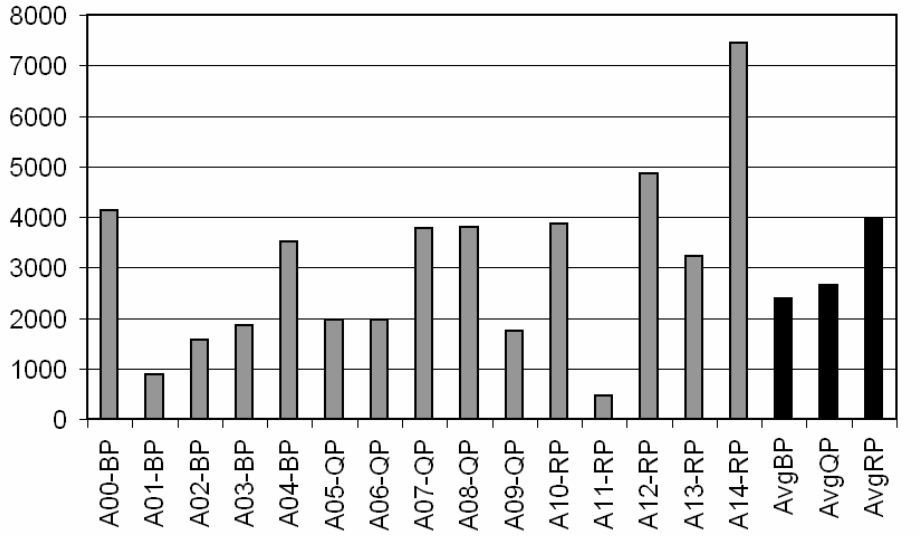

Figure 4. System behaviour when A00 agent rejects tasks with more than 169 training instances and more than 4 attributes

4.2.2. Strategy Based on Neural Networks

The above rule, however efficient, seems to be too simple, since it only takes into account 2 attributes of the internal learning problem. There are high chances that other factors or subtle interferences between them affect the optimal strategy. Therefore, a neural network was trained with the execution data. The attributes taken into account were: the number of attributes, the number of training instances, the number of testing instances, the type of the problem, and the number of competitors (other available agents in the system). The output is given by gaining or losing a reward.

The only difference from the application of C4.5 is that in this case the attributes were given numerical values, e.g. the type of the problem was 0 for regression and 1 for classification, and the output was 0 for “lost” and 1 for “paid”. Consequently, the neural network had a topology with 5 inputs, 1 hidden layer with 20 neurons, and 1 output. The confusion matrix after training is given in Table 7.

Table 7

Confusion matrix

	0 / lost	1 / paid
0 / lost	75.556 %	24.444 %
1 / paid	10.417 %	89.583 %

Now the trained neural network is used by agent A00 to accept or reject a task. The resulting behaviour of the multi-agent system is displayed in Figure 5.

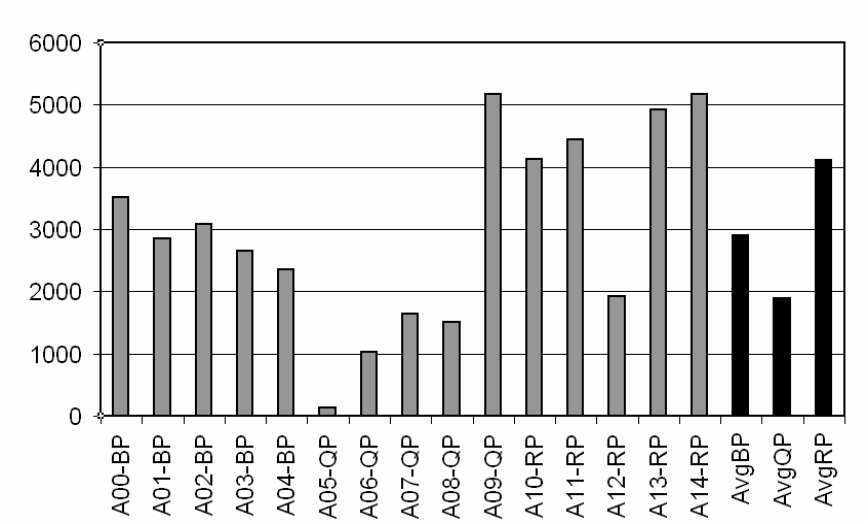

Figure 5. System behaviour when A00 agent uses a neural network to accept or reject tasks

One can see that the total utility of A00 is greater than any total utility of both Backpropagation and Quickprop agents. Moreover, the average total utility of the Backpropagation agents becomes larger that the average total of the Quickprop agents.

4.2.3. Global Effects of Optimization Strategy

The optimization of the strategy of an agent has clear effects on the performance of the agent. The next step was to extend the application of this optimization to all the Backpropagation agents. Figure 6 displays the resulting behaviour of the system when Backpropagation agents use the heuristic presented in paragraph 4.2.1, and Figure 7 displays the resulting behaviour of the system when Backpropagation agents use the neural network trained based on their execution experience.

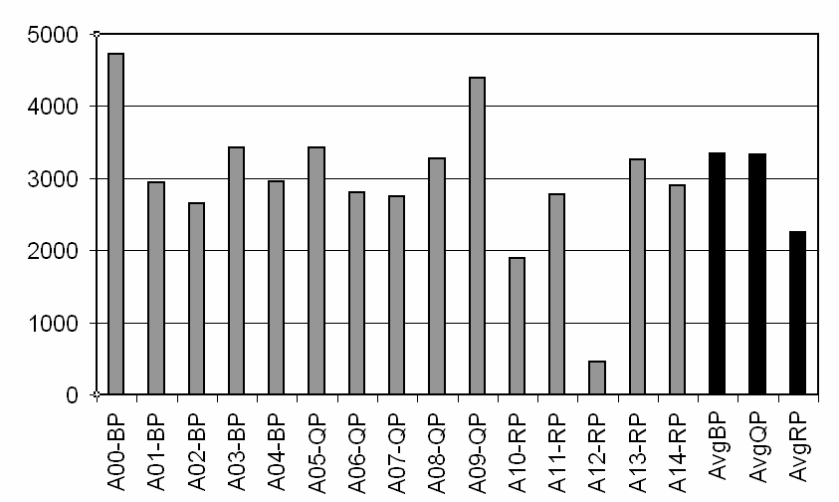

Figure 6. System behaviour when Backpropagation agents reject tasks with more than 169 training instances and more than 4 attributes

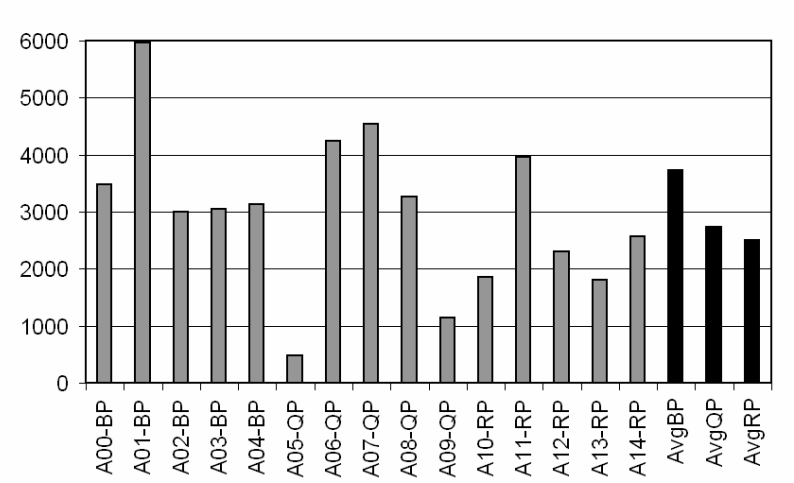

Figure 7. System behaviour when Backpropagation agents use a neural network to accept or reject tasks

It can be clearly seen that the overall performance of the Backpropagation agents significantly improves compared to that of other agents. Their performance is better when they use the neural network. Besides, this increase in performance also affects the other types of agents in the system. In the second case, the Backpropagation agents receive an average total utility greater than that of Quickprop and even Rprop agents.

5. Conclusions

From the individual tests on 20 classification and regression problems, the best training algorithm for neural networks seems to be Rprop, compared to Backpropagation and Quickprop, in terms of both execution speed and error rate.

By the analysis of the behaviour of the multi-agent system that contains agents which apply the three training algorithms, it was proven that the overall performance of an agent or of a group of agents can be improved by modifying the acceptance strategy of tasks.

By using an optimized strategy for task acceptance or rejection, the Backpropagation agents perform better on average than both Quickprop and Rprop agents. This result is surprising because the performance of the Backpropagation algorithm itself remains the same: it is usually inferior to the other two algorithms. The difference is made by the "intelligence" of accepting only the tasks that seem to fit the reduced capability of Backpropagation agents, and this is accomplished only by merging the common "group" experience, without any other type of cooperation or communication.

A very important point to be noted is that agents use neural networks to solve problems provided by the user, but also to learn from their own experience and consequently improve their behaviour. They use the same type of model for both external purposes and internal ones.

Aknowledgements

This work was supported by CNCSIS-UEFISCSU, project number PNII-IDEI 316/2008, *Behavioural Patterns Library for Intelligent Agents Used in Engineering and Management*.

References

Atanasiu, G.M., Leon, F., Popa, B.F. (2008), "Seismic Risk Mitigation in Urban Areas Based on Artificial Intelligence Methods", *Proceedings of the 14th World Conference on Earthquake Engineering*, Beijing, China

Bellifemine, F.L., Caire, G., Greenwood, D. (2007), *Developing Multi-Agent Systems with JADE*, Wiley Series in Agent Technology

Bryson, A.E., Ho, Y. C. (1969), *Applied Optimal Control*, Blaisdell, New York

Chavez, A., Maes, P. (1996), "Kasbah: An agent marketplace for buying and selling goods", *Proceedings of First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems*, London, UK

Crabtree, B., Jennings, N.R., eds. (1996), *Proceedings of First International Conference on the Practical Application of Intelligent Agents and Multi-agent Systems*, London, UK

Duda, R.O., Hart, P.E. (1973), *Pattern Classification and Scene Analysis*, John Wiley & Sons, p. 218

Fahlman, S.E. (1988), "Faster-Learning Variations on Back-Propagation: An Empirical Study", *Proceedings of the 1988 Connectionist Models Summer School*, Morgan-Kaufmann, Los Altos CA

Ferguson, I.A., Wooldridge, M. J. (1997), "Paying Their Way: Commercial Digital Libraries for the 21st Century", *D-Lib Magazine*, June 1997, available at:
http://www.dlib.org/dlib/june97/zuno/06ferguson.html

Fisher, R.A. (1936), "The use of multiple measurements in taxonomic problems", *Annual Eugenics*, 7, Part II, 179-188

Frank, E., et al. (2008), *Attribute-Relation File Format (ARFF)*, available at:
http://www.cs.waikato.ac.nz/~ml/weka/ arff.html.

Janca, P.C. (1995), *Pragmatic application of information agents*, BIS Strategic Report

Jennings, N.R., Wooldridge, M.J., eds. (1998), *Agent Technology: Foundations, Applications, and Markets*, Springer

Jennings, N., Faratin, P., Johnson, M. O'Brien, P., Wiegand, M. (1996), "Using intelligent agents to manage business processes", *Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology* (PAAM96), London, UK

Joshi, K.P. (1997), *Analysis of Data Mining Algorithms*, available at:
http://userpages.umbc.edu/~kjoshi1/data-mine/ proj_rpt.htm

Kudrycki, T.P. (1988), *Neural Network Implementation of a Medical Diagnostic Expert System*, Master's Thesis, College of Engineering, University of Cincinnati

Lashkari, Y., Metral, M., Maes, P. (1994), "Collaborative Interface Agents", *Proceedings of the 12th National Conference on Artificial Intelligence* (AAAI 1994), pp. 444-449, Seattle, WA, USA, AAAI Press

Leon, F. (2010), "Design of a Multi-agent System for Solving Search Problems", *Journal of Engineering Studies and Research*, Bacău, volume 16, no. 3, pp. 51-64

Leon, F., Lisa, C., Curteanu, S. (2010), "Prediction of the Liquid Crystalline Property Using Different Classification Methods", *Molecular Crystals and Liquid Crystals*, vol. 518, pp. 127–146

Leon, F., Zaharia, M.H., Gâlea, D. (2004), "Performance Analysis of Categorization Algorithms", *Proceedings of the 8th International Symposium on Automatic Control and Computer Science*, Iasi

Maes, P. (1994), "Agents that reduce workload and information overload", *Communications of the ACM*, vol. 37(7), July

Navarro, D.J., Myung, J. I., Pitt, M. A. (2004), "Comparing the qualitative performance of the ALCOVE and RULEX models of category learning", *37th Annual Meeting of the Society for Mathematical Psychology*, University of Michigan, Ann Arbor, MI

Nikos, V. (2007), "A Concise Introduction to Multi-agent Systems and Distributed Artificial", *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Vol. 1

Peer, E.S. (2005), *Computational Intelligence*, available at:
http://upetd.up.ac.za/thesis/available/etd-06102005-095510/unrestricted/ 02chapter2.pdf

Quinlan, J.R. (1986), "Induction of Decision Trees", *Machine Learning*, vol. 1, pp. 81-106

Quinlan, R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, CA

Riedmiller, M. (1994), *Rprop – Description and Implementation Details*, Technical Report, University of Karlsruhe